TRAVAUX PRATIQUES 5

Ce TP porte sur le chapitre « Programmation orientée objet et générique ».
Exercices complémentaires sur le polymorphisme, l'héritage et leur utilisation.

Créer un projet "TP5" dans Code::Blocks de type "Console - C++" (cf. TP1).

Nous allons créer une classe abstraite **Animal** qui sera dérivée en deux classes filles : **Chien** et **Chat**.

Exercice 1:

L'implémentation de la classe **Animal** est constituée de :

- Un entier `age`
- Un double `taille`
- Une chaîne de caractère `nom` initialement vide (pour les animaux sans noms)
- Une chaîne de caractère `proprietaire`
- Une fonction `fiche_info` retournant une chaîne de caractères ne prenant pas d'argument

L'interface de la classe contient :

- Le constructeur par défaut
- Un constructeur ayant autant d'argument que de données membres
- Le destructeur
- Une fonction `communication` retournant une chaîne de caractères et prenant une chaîne de caractère
- Une surcharge de la fonction `communication` prenant un pointeur de chaîne de caractères

(a) Décider du degré d'association de la classe **Animal** avec les chaînes de caractères de son implémentation. Commencer la déclaration de cette classe avec ces données membres.

(b) Constructeur par défaut et destructeur peuvent-ils employer la version "par défaut", justifier, compléter la déclaration en étant explicite dans ce choix.

(c) En considérant l'héritage à venir, nous décidons que :

- La fonction `communication` sera virtuelle pure
- Sa surcharge sur le type pointeur ne sera pas virtuelle
- La fonction `fiche_info` sera virtuelle

Réfléchir à ces choix et les expliquer. Compléter la déclaration.

(d) Toujours en considérant l'héritage, la donnée membre `proprietaire` et la fonction membre `fiche_info` devront être accessibles par les classes dérivées et elles seules, décider des niveaux d'encapsulation de toutes les données et fonctions membres. Compléter ou corriger la définition.

(e) On souhaite ajouter l'opérateur d'affichage `operator<<` à notre classe, qui sera disponible sur le type **Animal** et sur son pointeur. Compléter la définition en considérant qu'il faudra certainement que cette fonction accède à `fiche_info`.

(f) Quelles sont les fonctions ne modifiant pas l'état de la classe ? Le faire apparaître explicitement dans la définition d'**Animal**. Prendre aussi soin du passage des arguments.

Exercice 2:

(a) Écrire la définition des constructeurs.

(b) Écrire la définition des fonctions `communication`, la surcharge sur pointeur appelle simplement la version prenant une valeur.

(c) La fonction `fiche_info` retourne une chaîne de caractère comprenant :

— Le nom de l'animal s'il en a, sinon la chaîne « Sans nom »

— L'âge de l'animal

— Sa taille

— Le propriétaire s'il existe, sinon la chaîne « Sans propriétaire »

Tous ces champs apparaîtront chacun sur une ligne.
Écrire la définition de `fiche_info`.

(d) Écrire la définition des deux fonctions `operator<<`, elles appelleront la fonction `fiche_info`.

(e) Est-il possible de manipuler une instance de la classe **Animal** dans `main.cpp` ? Justifier.

Exercice 3:

La classe **Chien** ne contient pas de donnée membre complémentaire à **Animal**. La classe **Chien** utilise l'héritage sur **Animal** permettant d'exploiter le polymorphisme

— Les versions "par défaut" des fonctions pouvant être utilisées le seront explicitement.

— Les redéfinitions des fonctions virtuelles utiliseront le qualificateur `override` [1].

(a) La fonction `fiche_info` est-elle différente de celle héritée ? Si oui, la version **Chien** appellera la version héritée et la complétera.

(b) Quelles sont les autres fonctions membres nécessitant d'être déclarées ? Écrire la déclaration de la classe **Chien**.

(c) Écrire la définition des constructeurs et destructeur nécessaires.

(d) Écrire la définition des fonctions virtuelles le nécessitant. La fonction `communication` respectera l'algorithme suivant :

— L'argument de type chaîne de caractère représente « l'interlocuteur »

— Si l'instance de **Chien** possède un propriétaire et celui-ci est l'interlocuteur (simple test d'égalité de chaîne de caractères) alors la fonction retourne « Ouaf ! »

— Sinon « Grrr ! »

(e) Est-il nécessaire de proposer un `operator<<` pour le type Chien ? Justifier.

Exercice 4:

Dans le fichier `main.cpp` :

(a) Créer deux pointeurs de chaînes de caractères `moi` et `autre` contenant chacun le nom d'un propriétaire.

(b) Créer un pointeur d'**Animal** `pa` vide.

(c) Générer une instance de **Chien** dans le pointeur `pa` en utilisant le constructeur "par défaut".

— Utiliser l'opérateur d'affichage sur `pa`.

— Afficher le retour de la fonction communication avec `moi` comme argument.

— Même chose mais avec la deréférence de `moi`.

— Afficher le retour de la fonction communication avec `autre` comme argument.

(d) Même chose mais en générant une instance de **Chien** dans `pa` avec des valeurs pour chaque donnée membre, le propriétaire sera `moi`.

1. https://en.cppreference.com/w/cpp/language/override

Exercice 5:

La classe **Chat** contient une donnée membre supplémentaire `bonne_humeur` représentée par un booléen, par défaut fausse.

(a) Quelles sont les fonctions virtuelles à redéfinir ? Justifier et écrire la déclaration de la classe **Chat**.

(b) La fonction `fiche_info` est-elle différente de celle héritée ? Si oui, la version **Chat** appellera la version héritée et la complétera.

(c) Effectuer le même travail de définition sur la classe **Chat** que sur **Chien** avec les différences suivantes sur la fonction `communication` :

- Si l'instance de **Chat** possède un propriétaire, qu'il est l'interlocuteur et que le chat est de bonne humeur : « Rooooon »
- Si l'instance de **Chat** possède un propriétaire, qu'il est l'interlocuteur et que le chat est de mauvaise humeur : « .... »
- Si l'instance de **Chat** possède un propriétaire, qu'il n'est pas l'interlocuteur et que le chat est de bonne humeur : « Miaou »
- Si l'instance de **Chat** ne possède pas de propriétaire et que le chat est de bonne humeur : « Miaou »
- Si l'instance de **Chat** ne possède pas de propriétaire et que le chat est de mauvaise humeur : « Pssss »

Exercice 6:

Effectuer sur la classe **Chat** les mêmes tests dans le fichier `main.cpp` qu'avec la classe **Chien** pour un pointeur `pa` pointant sur :

- Un **Chat** construit "par défaut".
- Un **Chat** construit avec tous les arguments et de bonne humeur.
- Un **Chat** construit avec tous les arguments et de mauvaise humeur.

Exercice 7:

Au sujet des opérations de copie de ces trois classes :

(a) Vous interroger sur le degré de profondeur de copie nécessaire.

(b) En fonction de votre réponse, est-il nécessaire de définir une version spécifique des fonctions de copie ?

(c) Compléter déclaration et définition des classes selon vos choix en étant explicite si nécessaire.